Example

Code:

Console Application Code:

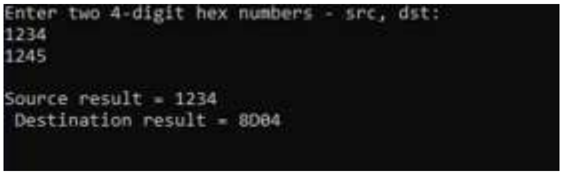
```
#include <stdio.h>
extern "C" void clear();
int main()
{
  clear();
  unsigned short src_opnd, dst_opnd, src_rslt, dst_rslt;
  printf("Enter two 4-digit hex numbers - src,dst: \n");
  scanf_s("%hX %hX", &src_opnd, &dst_opnd);
  _asm
  {
    MOV AX, src_opnd
    MOV BX, dst_opnd
    SHRD BX, AX, 10; shift AX : BX right 10 bits
    MOV src_rslt, AX
    MOV dst_rslt, BX

  }
printf("\nSource result = %X\n Destination result + %X\n\n", src_rslt, dst_rslt);
return 0;

}
```

ASM Code

```
686
.MODEL FLAT, C
.STACK 2048
.DATA

var_1 dword 10
str_1 byte 50,100,34,5,6,78,12,45,67
str_2 byte 5000 dup(?)
.CODe



clear PROC
   xor eax,eax
   xor ebx,ebx
```

```
ret
clear ENDP
END
```

Output



```
Enter two 4-digit hex numbers - src, dst:
1234
1245

Source result = 1234
 Destination result = 8D04
```

Task#1

```
#include <stdio.h>

// extern "C" instruct the compiler to use C calling conventions

extern "C" void Threeprod();

int main()

{         //define variables

          unsigned long f_opnd= 0, s_opnd =0, t_opnd =0 , dst_rslt= 0;

          printf("Enter Three 4-digit hex numbers - src, dst: \n");

          scanf_s("%hX %hX %hX", &f_opnd, &s_opnd, &t_opnd); // in scanf_s it is necessary to

          //specifiy length

          //switch to assembly

          _asm

          {

                    mov eax ,0

                    mov ebx ,0
```

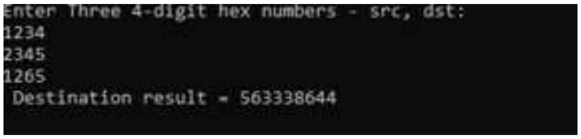
```
        mov ecx ,0


        MOV EAX, f_opnd

        MOV EBX, s_opnd

        MOV ECX, t_opnd

    }

    Threeprod();

    _asm

    {

        MOV dst_rslt, eax

    }

    printf(" Destination result = %d\n\n",  dst_rslt);

    return 0;

}
```

ASM code:

```
.686 ;Target processor. Use instructions for Pentium class machines

.MODEL FLAT, C ;Use the flat memory model. Use C calling conventions

.STACK 2048 ;Define a stack segment of 1KB (Not required for this example)

.DATA ;Create a near data segment. Local variables are declared after

;this directive (Not required for this example)

var_1 dword 10

str_1 byte 50,100,34,5,6,78,12,45,67

str_2 byte 5000 dup(?)

.CODE ;Indicates the start of a code segment.

Threeprod PROC

mul ebx

mul ecx

ret

Threeprod ENDP
```

```
END
```

OUTPUT:



```
Enter Three 4-digit hex numbers - src, dst:
1234
2345
1265
 Destination result = 563338644
```

Task#2

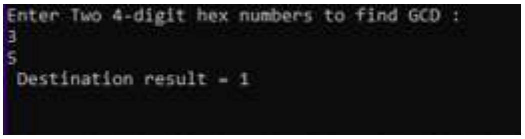
```
#include <stdio.h>

// extern "C" instruct the compiler to use C calling conventions

extern "C" void gcd_re();

int main()

{


        //define variables

        unsigned long f_opnd= 0, s_opnd =0, dst_rslt= 0;

        printf("Enter Two 4-digit hex numbers to find GCD : \n");

        scanf_s("%hX %hX", &f_opnd, &s_opnd); // in scanf_s it is necessary to

        //specifiy length

        //switch to assembly

        _asm

        {

                push f_opnd

                push s_opnd

        }


        gcd_re();


        _asm
```

```
    {
        MOV dst_rslt, eax
    }
    printf(" Destination result = %d\n\n",  dst_rslt);
    return 0;
}
```

ASM code:

```
.686 ;Target processor. Use instructions for Pentium class machines
.MODEL FLAT, C ;Use the flat memory model. Use C calling conventions
.STACK 2048 ;Define a stack segment of 1KB (Not required for this example)
.DATA ;Create a near data segment. Local variables are declared after

;this directive (Not required for this example)

var_1 dword 10
str_1 byte 50,100,34,5,6,78,12,45,67
str_2 byte 5000 dup(?)
.CODE ;Indicates the start of a code segment.



gcd_re PROC
    push ebp
    mov ebp, esp
    mov eax, [ebp + 12]          ; load first argument to EAX
    mov ebx, [ebp + 8]           ; load second argument to EBX
    mov edx, 0                   ; set EDX to 0 in order to divide
    div ebx
    mul ebx
    mov ebx, eax
    mov eax, [ebp + 12]
    mov edx, [ebp + 8]
    sub eax, ebx
    cmp eax, 0
    je L2
    push edx
    push eax
    call gcd_re

    L1:
        pop ebp
        ret 8

    L2:
        mov eax, edx
        jmp L1

gcd_re ENDP

end
```

OUTPUT:



```
Enter Two 4-digit hex numbers to find GCD :
3
5
 Destination result = 1
```